VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVAČ ŘEČI ŘÍZENÝ GRAMATIKAMI

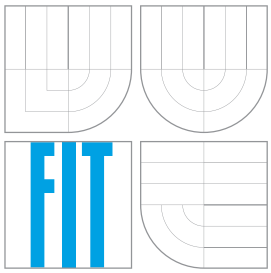
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH ŠKORVAGA

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ROZPOZNÁVAČ ŘEČI ŘÍZENÝ GRAMATIKAMI

GRAMMAR BASED AUTOMATIC SPEECH RECOGNIZER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. VOJTĚCH ŠKORVAGA
AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR SCHWARZ, Ph.D.
SUPERVISOR

BRNO 2014

Zadání diplomové práce/16337/2013/xskorv00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2013/2014

Zadání diplomové práce

Řešitel: **Škorvaga Vojtěch, Bc.**
Obor: Management a informační technologie
Téma: **Rozpoznávač řeči řízený gramatikami**
Grammar Based Automatic Speech Recognizer
Kategorie: Zpracování řeči a přirozeného jazyka

Pokyny:

1. Prostudujte formát gramatiky SGRS, rozpoznávač firmy Phonexia (aplikační rozhraní a formát rozpoznávacích sítí), způsob generování rozpoznávacích sítí pomocí konečných převodníků, síťový protokol MRCP v2 a open source projekt UniMRCP.
2. Napište algoritmus na převod SGRS na konečný automat.
3. Dle příkladu naimplementujte algoritmus na složení rozpoznávací síť na základě konečného automatu s gramatikou, převodníku popisujícího lexikon, převod na kontextově závislé fonémy, a na stavy.
4. Otestujte kompilátor gramatik v součinnosti s rozpoznávačem.
5. Napište modul pro projekt UniMRCP, který zapouzdřuje rozpoznávač řeči a přidává komunikaci s rozpoznávačem pomocí síťového protokolu MRCP v2.
6. Otestujte celkové řešení za pomoci některé IVR platformy (Asterisk, FreeSwitch, OptimSys)

Literatura:

- Dle konzultace se školitelem

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Schwarz Petr, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2013
Datum odevzdání: 28. května 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce popisuje vytvoření systému pro sestavení rozpoznávací sítě pro rozpoznávač mluvené řeči na základě Speech Recognition Grammar Specification (SRGS) gramatiky definované W3C konsorciem. Rozpoznávač byl spolu s tímto modulem integrován do softwarové ústředny FreeSWitch pomocí kombinace síťových protokolů MRCPv2/SIP/RTP a testován. Práce byla řešená ve spolupráci s firmou Phonexia s.r.o.

Abstract

This work describes a development of system for network compilation for speech recognition based on Speech Recognition Grammar Specification (SRGS) grammar defined by W3C consortium. Together with the new module, the recognizer was integrated to the FreeSwitch software phone switch using a combination of MRCPv2/SIP/RTP networks protokols and tested.

Klíčová slova

SRGS, OpenFST, Phonexia s.r.o., UniMRCP, MRCP, IVR, FreeSwitch

Keywords

SRGS, OpenFST, Phonexia s.r.o., UniMRCP, MRCP, IVR, FreeSwitch

Citace

Vojtěch Škorvaga: Rozpoznávač řeči řízený gramatikami, diplomová práce, Brno, FIT VUT v Brně, 2014

Rozpoznávač řeči řízený gramatikami

Prohlášení

Prohlašuji, že jsem tuto práci k semestrálnímu projektu vypracoval samostatně pod vedením pana Ing. Petra Schwarze, Ph.D.

.....
Vojtěch Škorvaga
28. května 2014

Obsah

Seznam obrázků

Kapitola 1

Úvod

Řeč je prostředek k interakci mezi lidmi již stovky tisíc let. Pomohla lidstvu sdělovat myšlenky, tvořit jazyky, kultury i národy. Přestože obraz vydá za tisíce slov, je to řeč, kterou ho můžeme popsat jinému člověku. A nyní, mohou lidé řečí komunikovat i se stroji. Stroje mohly mluvit na člověka již delší dobu, třeba díky jednoduchému pouštění audio záznamu. Opačný směr až v poslední době, díky růstu výpočetního výkonu a pokroku v analýze zpracování přirozeného jazyka.

V tomto textu se budu věnovat rozšíření rozpoznávače řeči od firmy Phonexia s.r.o. o prostředky o možnost definice rozpoznávaných vět pomocí W3C Speech Recognition Grammar Specification (SRGS) gramatik následně bude rozpoznávač a nový UniMRCP modul integrován do softwarové ústředny FreeSwitch pomocí protokolů MRCPv2/SIP/RTP. pro použití rozpoznávače řeči, který je součásti BSAPI, v řečových automatech postavených nad open source projekty jako UniMRCP a FreeSWITCH.

Cílem projektu je vytvořit funkční dialogový systém s možností popisu rozpoznávaných vět pomocí SRGS gramatiky. K tomu zapotřebí vytvořit převodník SRGS gramatik na konečný automat, kompilaci rozpoznávací sítě a vytvořit plugin pro UniMRCP projekt s využitím rozpoznávače firmy Phonexia.

Kapitola 2

Teoretická východiska práce

V této kapitole je popsán vztah SRGS (Speech Recognition Grammar Specification) k rozpoznávání řeči, hlasový dialogový systém (Interactive Voice Respose, IVR) a role rozpoznávače řeči (Automatic Speech Recognition, ASR) v tomto systému. Jde o obecné předpoklady pro vytvoření IVR systému s ASR řízeného pomocí SRGS gramatik pro společnost Phonexia.

2.1 Phonexia

Vyvíjí technologie pro automatické získávání informací z řeči (identifikace jazyka, řečníka, detekce klíčových slov a přepis řeči na text). Produkty se uplatňují při analýze telefonátů v call-centrech a v oblasti bezpečnosti nebo obrany.

Firma vyvíjí SDK (Software Development Kit) pro své produkty v C/C++, které pojmenovala BSAPI(Brno Speech API). Jeho součástí jsou technologie jako:

VAD (Voice Activity Detector) Systém pro detekci lidského hlasu v audiu.

SID (Speaker Identification) Systém pro identifikaci řečníka.

LID (Language Identification) Systém identifikace jazyka z audia.

KWS (Key Word Spotting) Systém pro vyhledávání klíčových slov v audiu.

STT (Speech To Text)Systém pro přepis řeči na text, také známý jako.

Tyto technologie mohou pracovat offline nebo online. Offline znamená, že se zpracovávají soubory s pořízenými nahrávkami, zatímco online zpracování znamená, že program čte data přímo z mikrofonu a pro analýzu nemůže využít zbytek promluvy, která ještě nebyla vyřčena.

Vstupem pro KWS a STT (souhrně ASR, Automatic Speech Recognition) je takzvaná rozpoznávací síť, která definuje co se bude hledat. V technologii KWS může tato síť obsahovat pouze hledaná klíčová slova, zatímco v STT obsahuje několik set tisíc až miliónu slov ze slovníku pro zvolený jazyk a popis jejich vazeb. Pro technologii KWS si tuto síť nadefinuje uživatel.

Rozpoznávací síť je datová struktur popisující konečný automat.

2.1.1 Tvorba rozpoznávacích sítí

Se označuje za kompilaci. Na začátku projektu nebyla tato funkce součástí rozpoznávacího jádra (BSAPI). Tuto funkcionalitu zajišťovala sada externích skriptů pro linuxovou příkazovou řádku, s využitím několika programů, pro převod různých formátu a pro manipulaci s konečnými automaty Skripty byly dodávány zákazníkům, ale jsou značně omezující.

Hlavní nevýhodou skriptové kompilace sítí je její platformní závislost. Linuxové skripty budou těžko fungovat na Windows. Díky tomu je nelze integrovat do systémů zákazníka. Dále je velmi omezující proces kompilace z uživatelského hlediska, probíhá následovně:

1. Spuštění.
2. Načtení slovníků spojení, které má síť rozpoznávat.
3. Načtení akustických modelů pro zvolený jazyk. Velikost souboru s modelem se pohybuje kolem 200 MB.
4. Kompilace síť pomocí OpenFST[1]. Tato část je inspirovaná projektem KALDI[2] jehož popis je v článku [4].
5. Uložení sítě do souboru.
6. Načtení sítě ze souboru do BSAPI.

Tento proces se opakuje pro každou uživatelem navrženou síť. Data se mezi jednotlivými fázemi kompilace předávají pomocí souborů. Proto je proces kompilace značně pomalý a těžkopádný.

K jeho ergonomii nepřispívá ani metoda zadávaní slovníkových spojení, které má rozpoznávač rozpoznávat. Její podstatou je výčet všech možností. Při takovém způsoby zápisu by měl vstupní soubor pro rozpoznaní hesla, které může mít 12 číslic, asi $1e^12$ řádků.

Z těchto důvodů je cílem této diplomové práce vytvořit v BSAPI kompilátor rozpoznávacích sítí, který jako vstup bude akceptovat gramatiku z kapitoly definovanou W3C SRGS.

2.2 Rozpoznávač řeči

Zachycuje obrázek 2.1. Z řeči jsou extrahovány příznaky. K těm se potom hledá odpovídající index do tabulky, která tvoří akustický model. Tyto informace se posílají do dekodéru, který předtím načetl rozpoznávací síť.

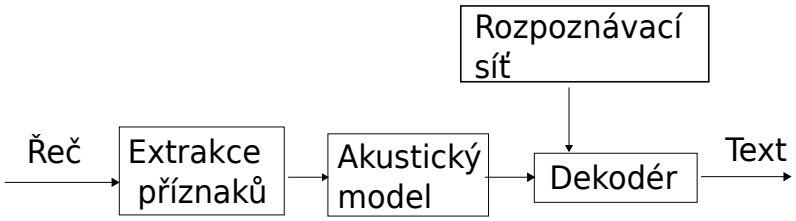
2.2.1 Dekodér

Pracuje, tak že na základě rozpoznávací sítě a výstupu z porovnání akustického modelu s řečí tvoří hypotézy. Které dále zpřesňuje a eliminuje nepravděpodobné průchody rozpoznávací síti.

Samotný text dekodér získá tak že z porovnání akustického modelu s řečí získá indexy do tabulky akustického modelu. Tyto indexy reprezentují kontextově závisle fonémy, které se

[1] OpenFST je knihovna pro reprezentaci konečných automatů. Domovská stránka projektu je na webu http://www.openfst.org/twiki/bin/view/FST/WebHome

[2] Představení toolkitu na webu: http://www.fit.vutbr.cz/units/UPGM/prod/index.php?id=304&notitle=1



Obrázek 2.1: Schéma rozpoznávače řeči.

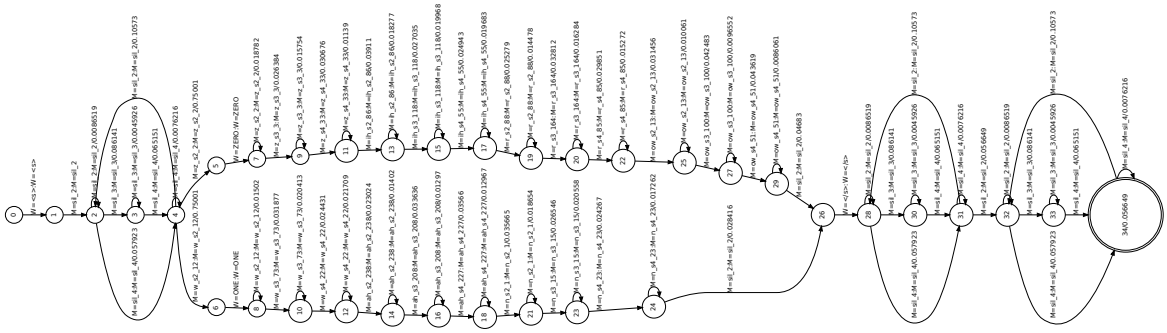
převedou na kontextově nezávislé. A posloupnost kontextově nezávislých fonému odpovídá slovu.

Dekodér pracuje s více možnými výsledky. Jako výsledek prohlásí ten s nejvyšší pravděpodobností.

2.3 Rozpoznávací síť

Se podle článku [7] skládá ze 3 převodníků (převodník vysvětlen v 2.3.3) a jednoho konečného automatu. Převodník[3] je graf který na svých hranách rozlišuje vstupní a výstupní řetězec. Průchod automatem do konečného stavu pak mapuje jednu větu na druhou. Jinak řečeno převede jeden vzor na druhý od toho název převodník.

Na obrázku 2.2 je rozpoznávací síť, která popisuje promluvu jednoho z anglických slov "one"nebo "zero". Která byla vytvořena podle podle článku [7]. Síť obsahuje akustické



Obrázek 2.2: Vizualizace rozpoznávací sítě pro anglické slovo "ONE"nebo "ZERO"v OpenFST.

jednotky a slova. Akustické jednotky jsou jsou na vstupech hran převodníků a slova na jejich výstupu.

[3] Anglicky transducer.

2.3.1 Definice konečného automatu

Zkráceně KA, je pětice:
$KA = (Q, \sum, \sigma, s, F)$, kde:
Q je konečná množina vnitřních stavů automatu.
$\sum$ konečná množina vstupních symbolů automatu, také nazývaná abeceda.
σ je přechodovou funkci ze stavu do stavu, definovanou jako $Q \times \{\sum \cup \varepsilon\} \to 2^Q$
s je množina počátečních stavů, taková že $s \subseteq Q$
F je množina koncových stavů, taková že $F \subseteq Q$

2.3.2 Definice váhovaný konečného automatu

Akceptor je vlastně konečný automat, ale pro potřeby této práce je ho musíme rozšířit o váhu z okruhu K. Okruh reprezentuje váhu s jakou se může uskutečnit přechod nebo akceptovat koncový stav.

Upravená definice tedy vypadá jako uspořádaná šestice: $A = (Q, \sum, \sigma, s, F, p)$, kde:
Q je konečná množina vnitřních stavů automatu.
$\sum$ konečná množina vstupních symbolů automatu, také nazývaná abeceda.
σ je množina přechodů, ve tvaru $Q \times \{\sum \cup \varepsilon\} \times K \to 2^Q$. s je množina počátečních stavů, taková že $s \subseteq Q$
F je množina koncových stavů, taková že $F \subseteq Q$
p je váhová funkce koncových stavů, taková že $p : F \to K$

2.3.3 Definice váhovaného převodníku

Váhovaný Převodník je váhovaný konečný automat, rozšířený o výstupní symbol na hraně. Akceptováním nějaké věty pak dojde k jejímu přepisu z jazyka definovaného konečným automatem a jeho vstupními symboly do jazyka definovaného konečným automatem a jeho výstupními symboly.

V článku [7] a [6] je převodník s váhami nad okruhem K definován jako osmice: $T = (\sum, \triangle, Q, I, F, E, \lambda, p)$, kde
$\sum$ je konečná abeceda vstupních symbolů.
$\triangle$ je konečná abeceda výstupních symbolů.
Q je konečná množina vnitřních stavů automatu.
I je množina počátečních stavů, taková že $I \subseteq Q$
F je množina koncových stavů, taková že $I \subseteq Q$
E je množina přechodů, taková že $E \subseteq Q \times (\sum \cup \{\epsilon\}) \times (\triangle \cup \{\epsilon\}) \times K \times Q$
λ je váhová funkce počátečních stavů, taková že $\lambda : I \to K$
p je váhová funkce koncových stavů, taková že $p : F \to K$

2.3.4 Vztah váhovaného konečného automatu a převodníků

Váhovaný konečný automat je současně speciálním typem váhovaného převodníku, který má na vstupní a výstupní hraně stejný symbol. Váhovaný konečný automat tedy lze jednoduše převést na převodník:

Data: Váhovaný konečný automat $A = (Q_A, \sum, \sigma, s, F_A, p_A)$
Result: Váhovaný převodník $T = (\sum, \sum, Q_A, s, F_A, E, (s, 1), p_a)$
method
foreach $(start_s, symbol, likelihood, end_s) \in \sigma$ **do**
| $E \leftarrow E \cup \{(start_s, symbol, symbol, likelihood, end_s)\}$
end

Algorithm 1: Převod váhovaného KA na váhovaný převodník.

2.3.5 Komponenty rozpoznávací sítě

Při skládaní rozpoznávací sítě se podle článku [7] využívá těchto váhovaných převodníků:

G konečný automat popisující gramatiku. Tento graf je výstupem z analýzy SRGS gramatiky.

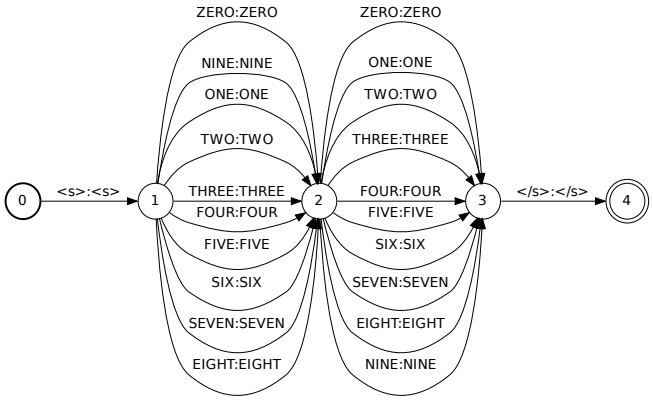
L lexikální převodník. Mapuje fonémy na slova. Fonémy jsou reprezentované třeba fonetickou abecedou.

C kontextový převodník. Mapuje fonémy kontextově fonémy.

H převodník mapuje kontextově závislé fonémy na stavy. Které jsou popsány indexy do tabulky stavů skrytých Markovových modelů (HMM[4])

Komponenta G

Je váhovaný konečným automatem reprezentující uživatelem zadanou SRGS gramatiku. Obrázek 2.3 ukazuje konečný automat pro gramatiku přijímající dvě číslovky. Na hranách automatu jsou vidět čísla definovaná uživatelem.

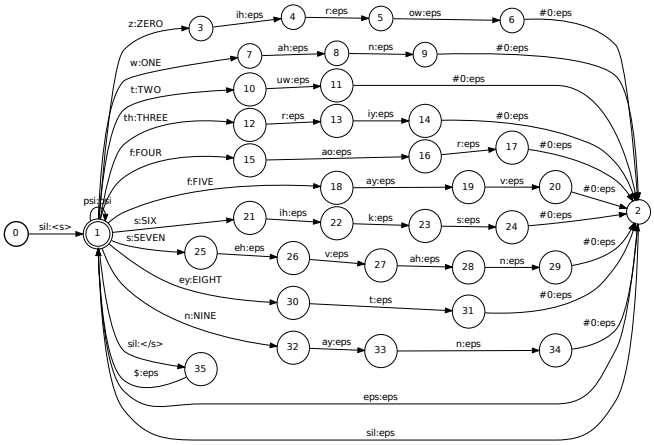


Obrázek 2.3: Vizualizace konečného automatu G, reprezentující gramatiku v OpenFST.

[4] Anglicky Hidden Markov Model.

Komponenta L

Reprezentuje výslovnost slov v konečném automatu G (gramatice), tím že převádí fonémy na slova. Na obrázku 2.4 jsou vidět fonémy na vstupech hran a slovo nebo symbol pro ε na výstupech hrany. Protože vychází z výslovnosti tak jedno slovo může mít více výslovností a tím pádem i více fonetických přepisů[5]. Nebo naopak rozdílná slova s jedním fonetickým přepisem, která jsou v češtině označována za homonyma. V případě přidání fonetického přepisů se problém řeší jednoduše pouhým vložením další větve ze stavu 1 do stavu 2. V případě homonym je to složitější. Aby bylo možno převodníkem pracovat, tak článek v [7] autor přidává na konec každého fonetického přepisu homonym pomocný symbol. Tento symbol nemají žádné dva stejně znějící homonyma stejný. Symbol začíná znakem **#** a pokračuje číslem. V pozdějších fázích sestavení rozpoznávací sítě se odstraňuje převedením na symbol ε, aby nepřekážel dekodéru.



Obrázek 2.4: Vizualizace převodníku L pro anglické číslovky v OpenFST.

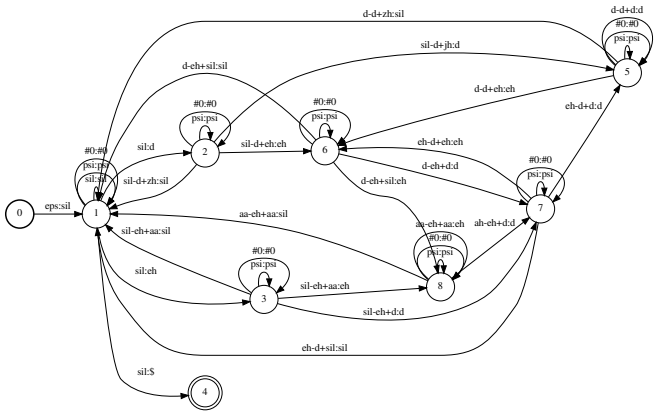
Komponenta C

Váhovaný převodník mapuje kontextové závislé fonémy na nezávislé fonémy. Využití kontextových fonému je motivováno zvýšením přesnosti rozpoznávače řeči. Převodník má na vstupu hrany kontextově závislý foném a na výstupu nezávislý fonem, které na vstupu hranách převodníků L. Příklad převodníků C pro tři fonémy je vidět na obrázku 2.5.

Komponenta H

Váhovaný převodník mapuje index do tabulky obsahující akustický model na kontextově závislé fonémy. Obrázek 2.6 zachycuje výřez převodníku H, který byl vytvořen pro kontextově závislé fonémy z obrázku 2.5. Na vstupu hran je vidět index a na výstupu je kontextově závislý foném.

[5]Fonetický přepis je prostý přepis slova na posloupnost fonému kterými je vyslovován.



Obrázek 2.5: Vizualizace převodníku C v OpenFST.

2.3.6 Skládaní rozpoznávací sítě

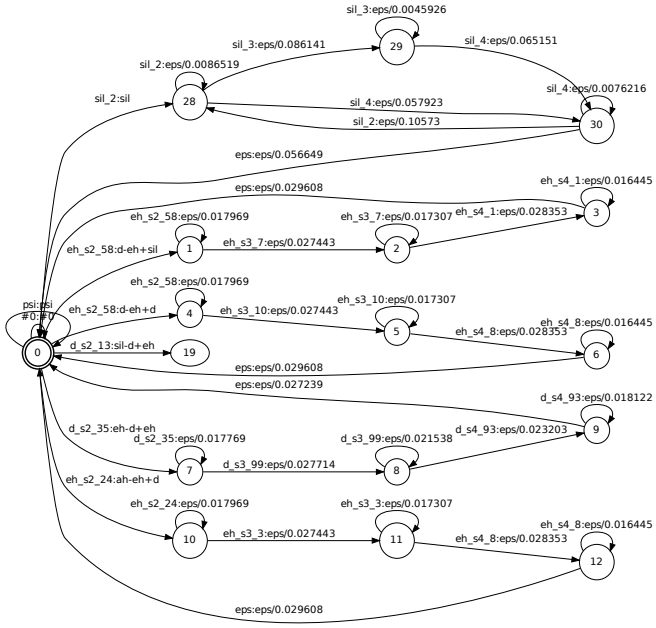
Začíná přečtením gramatiky. Pokud je gramatika bez chyb je převedena na automat G. Poté se dle článků [7] automat G složí s lexikálním převodníkem L, který je vygenerován ze slovníku přepisující slova na posloupnost fonémů. Dál jsou dle vzorce $HCLG = H \circ min\left(C \circ det\left(L \circ G\right)\right)$ připojeny váhované převodníky C a H. Kde $HCLG$ je rozpoznávací sítí.

Operace $\circ$ značí skládaní dvou váhovaných převodníků a je vysvětlena dále.

Skládání dvou převodníků s váhou

Operace skládání dvou výhovaných převodníku T_1 a T_2 je popsána v článku [7] a [6] pod anglickým pojmenováním "compose". Její princip je vyjádřen pseudo kódem 2. Kde:

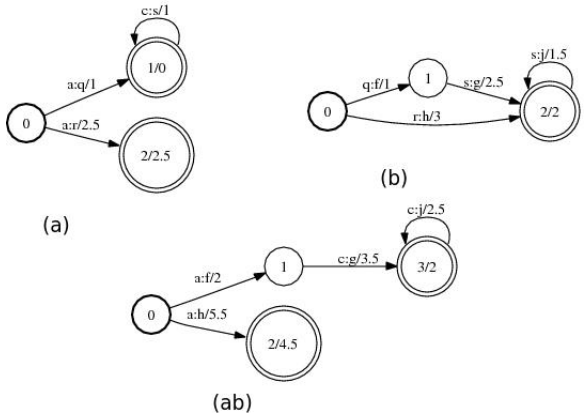
- *wight* je funkce mapující hranu na její váhu.
- $X\left[e\right]$ je funkce mapující hranu e na hodnotu z množiny X. Zjednodušeně lze říct, že se jedná o výběr z pole.
- $\otimes$ je násobení.



Obrázek 2.6: Vizualizace výřezu z převodníku H v OpenFST.

Data: $T_1 = (A, B_1, Q_1, I_1, F_1, E_1, \lambda_1, p_1)$
$T_2 = (B_2, C, Q_2, I_2, F_2, E_2, \lambda_2, p_2)$
Result: $T = (A, C, Q, I, F, E, \lambda, p)$
method
$Q \leftarrow I_1 \times I_2$
$S \leftarrow I_1 \times I_2$
while $S \neq \emptyset$ **do**
  $(q_1, q_2) \leftarrow HEAD(S)$
  $DEQUEUE(S)$
  if $(q_1, q_2) \in I_1 \times I_2$ **then**
    $I \leftarrow I \cup \{(q_1, q_2)\}$
    $\lambda(q_1, q_2) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
  end
  if $(q_1, q_2) \in F_1 \times F_2$ **then**
    $F \leftarrow F \cup \{(q_1, q_2)\}$
    $p(q_1, q_2) \leftarrow p_1(q_1) \otimes p_2(q_2)$
  end
  for *each* $(e_1, e_2) \in E[q_1] \times E[q_2]$ *such that* $B_1[e_1] = B_2[e_2]$ **do**
    if $(n[e_1], n[e_2]) \notin Q$ **then**
      $Q \leftarrow Q \cup \{(n[e_1], n[e_2])\}$
      $ENDQEUE(S, \{(n[e_1], n[e_2])\})$
    end
    $E \leftarrow E \cup \{((q_1, q_2), A[e_1], C[e_2], weight(e_1) \otimes weight(e_2), (n[e_1], n[e_2]))\}$
  end
end

Algorithm 2: Pseudo algoritmus pro skládaní dvou převodníku.

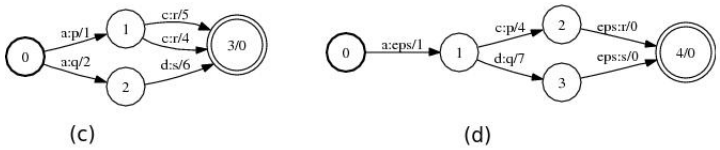


Obrázek 2.7: Příklad skládaní převodníků. (a) převodník A, (b) převodník B, (ab) složený převodník A∘B. Obrázky jsou přejaty z projektu OpenFST.

Determinizace převodníku

Podle článku [7] a [6] obsahuje jeden stav maximálně jednu hranu s každým vstupním symbolem ze vstupní abecedy a ε se nevyskytuje na vstupu žádné hrany. Obrázek 2.8 ukazuje příklad determinizace. Determinizace způsobí, že pro každý vstupní řetězec existuje jenom jedna možná cesta automatem. Tím se urychlí analýza řeči v ASR.

Determinizovaný váhovaný převodník musí dávat pro stejný vstupní řetězec stejný výstup se stejnou pravděpodobnosti. Ne váhovaný převodník, nebo váhovaný konečný automat může být determinizován.



Obrázek 2.8: Příklad determinizace převodníků. (c) převodník C, (d) determinizovaný převodník C. Obrázky jsou přejaty z projektu OpenFST.

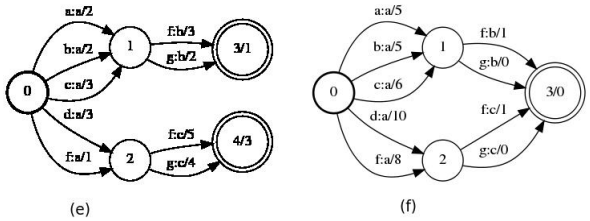
Minimalizace váhovaného převodníku

Se provádí aby byla zmenšena jeho velikost a tím byl ušetřen čas během analýzy řeči v ASR. Na obrázku 2.9 je vidět příklad minimalizace.

2.4 Gramatika pro rozpoznávání řeči

Anglicky Speech Recognition Grammar, zkratka SRG. Někdy označována jako SRGS, kde poslední S je pro anglické slovo Specification. Je specifikace sady pravidel pro informování řečového rozpoznávače o tom, co čekáme, že uživatel řekne. Standard vytvořila a udržuje skupina W3C(World Wide Web Consortium) [6].

[6] W3C je mezinárodní uskupení pro vývoj webových standardu, více na webu www.w3.org .



Obrázek 2.9: Příklad minimalizace převodníků. (e) převodník E, (f) minimalizovaný převodník E. Obrázky jsou přejaty z projektu OpenFST.

SRGS popisuje bezkontextovou gramatiku. Gramatika definovaná s pomocí SRGS je regulární gramatikou. Rozpoznávač je schopen pracovat pouze s regulární gramatikou definovanou konečným automatem na jehož hranách jsou slova a akustické jednotky, nebo ε pro reprezentaci žádného slova. Tento automat je uložen v rozpoznávací síti.

která je ekvivalentní s konečným automatem, na jehož hranách jsou očekávána slova nebo epsylon pro reprezentaci žádného slova.

Výstupem z rozpoznávače je pak řetězec slov odpovídající průchodu konečným automatem.

SRGS dovoluje gramatiky zapsat ve dvou ekvivalentních syntaxích:

- **ABNF** Rozšířena, anglicky Augmented, Backus/Naurova Forma[7].
- **XML** forma.

2.4.1 ABNF forma

Je syntaxe v čistém textu. ABNF je na rozdíl od XML zápis přehlednější a kratší. Díky tomu se s ním, novým uživatelům, lépe pracuje. Soubor v tomto formátu gramatiky by podle SRGS měl mít na svém začátku text "`#ABNF`"(bez uvozovek).

Příklad gramatiky pro objednání pizzy s pitím:

```
#ABNF 1.0 ISO-8859-1;
language en-US;
// root role is start point of gramar
root $pizza;
public $digit = one | two | three | four | five | six | seven | eight | nine;
public $number= $digit <1-> ;
public $kind = mushrum | salami | cheese | feferony;
public $drink = (big | small) colla | juice;
public $pizza = [ $number ] $kind [ pizza ] and [ $drink ];
```

2.4.2 XML forma

Existuje protože XML je obecným standardem pro předávaní dat. Jednotlivé prvky SRG jsou zapsány v XML elementech a jejich atributech. Soubor s touto formou gramatiky by podle SRGS měl mít na svém začátku "`#XML`"(bez uvozovek).

[7]ABNF je rozdílná od EBNF (Extended Backus/Naurova form). ABNF je definováno v RFC 5234 a EBNF v ISO 14977.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
                  "http://www.w3.org/TR/speech-grammar/grammar
                     .dtd">

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/06/grammar
␣␣␣␣␣␣␣␣␣http://www.w3.org/TR/speech-grammar/grammar.xsd"
         xml:lang="en-US" version="1.0"
         root="pizza">

<rule id="digit" scope="public">
      <one-of>
      <item>one</item>
      <item>two</item>
      <item>three</item>
      <item>four</item>
      <item>five</item>
      <item>six</item>
      <item>seven</item>
      <item>eight</item>
      <item>nine</item>
    </one-of>
</rule>
  <rule id="number" scope="public">
    <item repeat="1-"><ruleref uri="#digit"/></item>
  </rule>
  <rule id="kind">
    <one-of>
      <item>mushrum</item>
      <item>salami</item>
      <item>cheese</item>
      <item>feferony</item>
    </one-of>
  </rule>
  <rule id="drink" scope="public">
    <one-of>
      <item>
        <one-of>
          <item>big</item>
          <item>small</item>
        </one-of>
        colla
      </item>
      <item>juice<\/item>
```

```
    </one-of>
  </rule>
  <rule id="pizza" scope="public">
    <item repeat="0-1"><ruleref uri="#number"/></item>
    <item><ruleref uri="\#kind" \/><\/item>
    <item repeat="0-1"> pizza <\/item>
    and
    <item repeat="0-1"><ruleref uri="#drink"/></item>
  </rule>
</grammar>
```

2.4.3 Elementy SRGS

Tabulka 2.1 obsahuje přehled elementu SRGS jazyka.

Jméno	ABNF příklad	XML příklad	Popis
Pravidlo	`$jmeno =`	`<rule uri="#jmeno"/>`	
Alternativa	`A|B`	xml elementy na stejné úrovni	může být vysloveno A nebo B
Opakování	`<1-9>`	`<item repeat="1-9">`	určí počet možných opakování bloku

Uzavření do bloku	`(vol1 | vol2 )`	`<one-of>`
		`<item>vol1</item>`
		`<item>vol2</item>`
		`</one-of>`
			aby celý blok mohl být na straně pravidla —

Nepovinný blok	`[ vol1 ]`	`<one-of repeat="0-1">`	
Tag	{!{ }!}	`<tag> <tag>`	Rozšiřuje sémantiku[8]
Pravděpodobnost	`/0.7/`	`<itemrepeat-prob=".7">`	
Reference na pravidlo	`$jmeno`	`<ruleref uri="#jmeno">`	
Vstupní pravidlo	`root $jmeno;`	`<grammar root="jmeno">`	Identifikuje první volané pravidlo v gramatice.

Tabulka 2.1: Přehled základních elementů jazyka SRGS/ABNF

2.5 Hlasový dialogový systém

IVR slouží pro nahrazení člověka jako rozhodující jednotky v ústředně. Typické použití je když zavoláte na nějakou službu a z telefonního sluchátka se ozve pro objednávku zmáčkněte "1", pro reklamaci "2", a tak dále. Pokud je tato interakce vykonávána automatem namísto lidmi tak firmy ušetří peníze, což je hlavní důvod jejich používání.

Pokud je zákazník nucen pro vyjádření svého přání stisknout klávesu na telefonu je použita technologie DTFM[9], která převede stisk klávesy na specifický tón. Ten IVR rozezná a podle něj rozhodne o dalším postupu. Při použití ASR (Automatické rozpoznávání řeči), není uživatel nucen používat klávesnici a plynule pokračuje v rozhovoru. To přináší

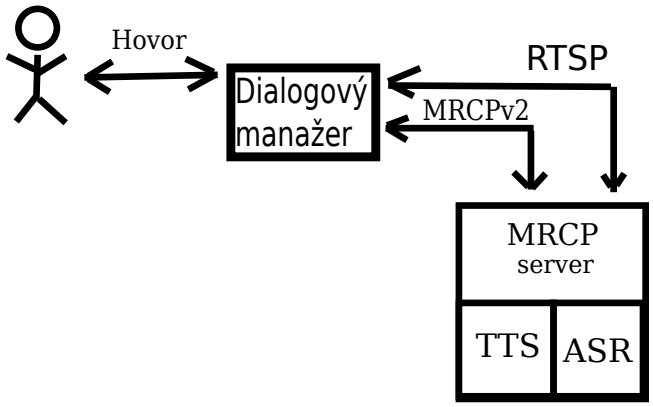
[9]Detaily o DTFM naleznete v RFC4733 na webu http://tools.ietf.org/html/rfc4733

výraznější spokojenost zákazníků (podle článku [5] až o 20%). Nemluvě o použití IVR jako vstupu pro hlasem ovládané prostředí.

Při automatickém rozhovoru může IVR přehrávat uživateli předpřipravené nahrávky nebo využít TTS (Text To Speech), převod textu na řeč.

2.5.1 Obecné schéma IVR systému

Je na obrázku 2.10. Uživatel vede hovor s dialogovým manažerem, který má nadefinovaný scénář, kterým uživatele provede. Po otevření telefonní linky muže přehrát předpřipravenou nahrávku nebo vygenerovanou pomocí TTS. Poté pomocí protokolu MRCP inicializuje ASR rozpoznávání. Další MRCP zpráva obsahuje SRGS element, kterým informuje ASR server o tom, co očekává, že uživatel řekne, co je validní uživatelovou odpovědí. Dialogový manažer přeposílá uživatelovou odpověď na ASR server, který po nalezení shody s SRGS v těle MRCP zprávy vrací rozpoznanou promluvu v textovém řetězci. V případě vypršení časového limitu, nebo-li nerozpoznání informuje dialogový manažer chybovým stavem. Ten poté zareaguje na odpověď podle svého scénáře.



Obrázek 2.10: Schéma IVR systému.

2.5.2 Existující řešení

2.6 Protokol MRCP

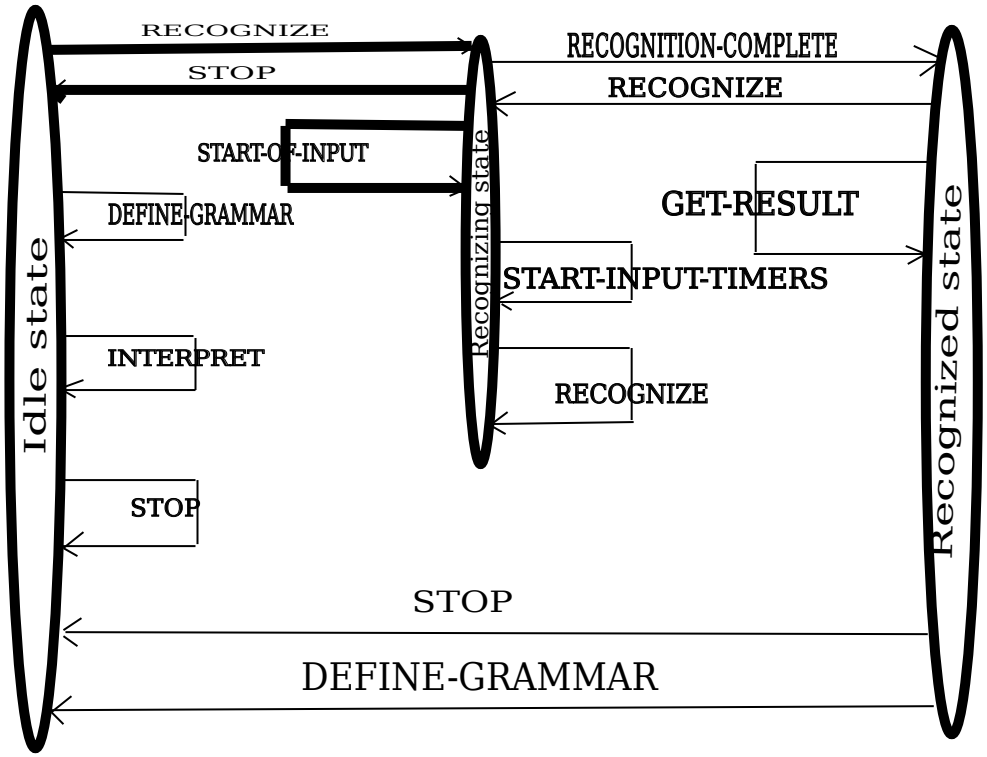
Media Resource Control Protocol [3]. Je textový, síťový protokol, který je určen pro komunikaci klientů s řečovými servery. MRCP klienti jím řídí průběh třeba rozpoznávaní řeči nebo syntézy řeči.

Spojení klienta a serveru se řeší pomocí SIP[10](Session Initiation Protocol) hlas se přenáší pomocí RTP[11](Real Time Streaming Protocol). Standardní protokoly dovolují implementaci systému do produktů třetích stran, jako je třeba FreeSWITCH[12] server pro IP telefonii.

[10]Více v RFC3261, dostupného na webu http://www.ietf.org/rfc/rfc3261.txt
[11]Více v RFC3550, dostupného z webu http://tools.ietf.org/html/rfc3550
[12]Opensource projekt s domovským webem na adrese http://www.freeswitch.org/

2.6.1 Proces rozpoznávání v MRCP

MRCP klient řídí proces rozpoznávání na serveru podle stavového automatu zobrazeného na obrázku 2.11. Na něm je patrné že rozpoznávač nemusí po úspěšném rozpoznání vzoru, stav "Recognized", skončit.



Obrázek 2.11: Stavový automat rozpoznávaní.

2.6.2 UniMRCP

UniMRCP[13] je open source projekt pro podporu vývoje MRCP serveru a MRCP klientu. Podporujícího MRCPv1 a MRCPv2. MRCPv1 používá k řízení sezení RTP a MRCPv2 SIP. Podporuje jak Linuxové tak Windowsové systémy.

Projekt je napsán v C/C++ s modulární strukturou. To dovoluje přidat novou funkcionalitu napsáním modulu, to obnáší vyplnění těla několika funkcí, které jsou definovány jako rozhraní.

UniMRCP projekt obsahuje rozhraní pro tyto MRCP aplikace:

- ASR

[13]Domovská stránka projektu je na http://www.unimrcp.org/

- TTS
- SR (Speech Recorder), nahrávaní řeči.
- SVI (Speaker Verifier and Identifier), ověření a identifikace řečníka.

Projekt se především zaměřuje na podporu imlementace MRCP klienta a nesnaží se vytvořit řečový server. UniMRCP obsahuje ukázkové implementace rozhraní pro MRCP aplikace.

Kapitola 3

Převod SGRS gramatik na váhovaný konečný automat

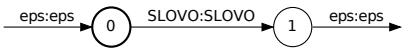
V této kapitole bude vysvětleno jak je SRGS/ABNF gramatika převedena na konečný automat reprezentovaný pomocí tříd z knihovny OpenFST. Ke čtení zapsané gramatiky je využita knihovna ANTLR[1] (ANother Tool for Language Recognition), která generuje programový kód v C/C++ pro lexikální a syntaktickou analýzu specifikovaného jazyka.

Pro implementaci převodu byla zvolena sada elementů SRGS/ABNF, která dovoluje základní práci s gramatikami. Ostatní prvky jazyka podle [2] jsou rozpoznány, ale nejsou použity. Uživateli je ovšem o tomto faktu předaná zpráva.

Seznam elementů SRGS/ABNF, které se projeví na tvaru výstupního váhovaného konečného automatu (VKA):

Slovo

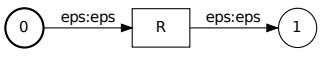
Jednotlivá rozpoznávána slova tvoří terminály v SRGS. Obrázek 3.1. V implementaci musí být zapsána velkými písmeny.



Obrázek 3.1: Schéma elementu slovo v VKA G.

Definice pravidla

S předpisem `$rulename=R;` se uloží pod jménem `rulename` do seznamu pravidel.



Obrázek 3.2: Schéma elementu pro definovaní pravidla v VKA G.

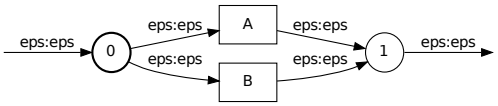
[1] Domovská stránka projektu je webu http://www.antlr.org/

Reference na pravidlo

s předpisem `$R`. Ze seznamu pravidel zkopíruje jeho graf a vloží ho na požadované místu definici nového pravidla.

Alternativa

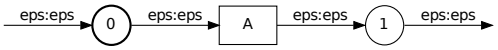
S předpisem `A|B` se do pravidla gramatiky vloží, tak že se vedou ε přechody od obou možností průchodů. Detail na obrázku 3.3. A a B představují podgrafy vytvořené ze zápisu gramatiky pro volbu A nebo B.



Obrázek 3.3: Schéma elementu pro výběr alternativy v VKA G.

Blok

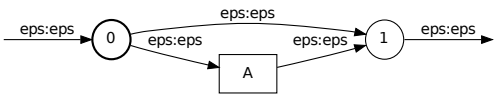
Uzavírá část A do bloku obaleného stavy, aby na něj bylo možno aplikovat elementy SRGS. Obrázek 3.4.



Obrázek 3.4: Schéma elementu pro vybrání jednu z možností nebo nic v VKA G.

Nepovinný blok

Blok rozšiřuje rozšiřuje o ε hranu, která ho dovoluje přeskočit. A tak vytvořit nepovinnou část gramatiky. Obrázek 3.5.

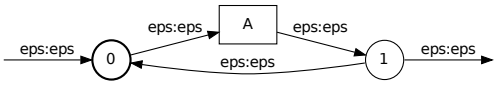


Obrázek 3.5: Schéma elementu pro vybrání jednu z možností nebo nic v VKA G.

Opakování

S předpisem `<1->` znamená, že část gramatiky se může do nekonečna opakovat. Realizuje zpětnou ε hranou. Obrázek 3.6. Jednička může být nahrazena jakýmkoliv kladným číslem.

Konkrétní počet opakování s předpisem `<4>` se realizuje tak, že se opakovaný blok nakopíruje za svůj konec.

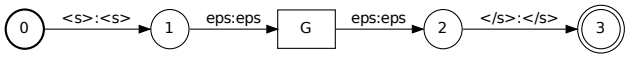


Obrázek 3.6: Schéma elementu pro nekonečné opakování v VKA G.

Rozsah s předpisem `<2-4>` je pak realizován kombinací konkrétního počtu opakování a nepovinných bloků.

Vstupní pravidlo

S předpisem `root=$A;`, vyhledá vstupní pravidlo do gramatiky, které rozšíří aby tvořilo plnohodnotný automat. Obrázek 3.7.



Obrázek 3.7: Schéma elementu pro začátek gramatiky v VKA G.

3.1 Vstup

Vstupem do převodu je SRGS/ABNF gramatika. Příklad gramatiky zpracování požadavku v pizzerii:

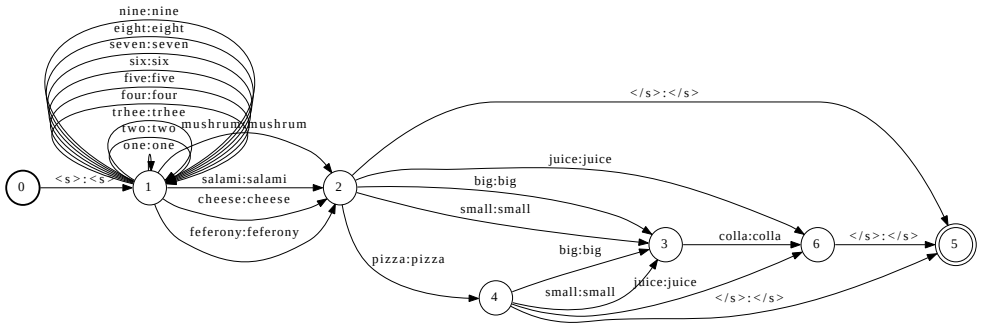
```
#ABNF 1.0 ISO-8859-1;
language en-US;
// root role is start point of gramar
root $pizza;
public $digit = one | two | three | four | five | six | seven | eight | nine;
public $number= $digit <1-> ;
public $kind = mushrum | salami | cheese | feferony;
public $drink = (big | small) colla | juice;
public $pizza = [ $number ] $kind [ pizza ] [ $drink ];
```

3.2 Výstup

Výstupem z převodu je váhovaný konečný automat. Obrázek 3.8 je výstupem k příkladu v předešlé kapitole 3.1.

3.3 Postup převodu gramatiky na graf G

K analýze SRGS/ABNF gramatiky je použit nástroj ANTLR podle knihy [10]. Po identifikaci každého z přijímaných elementu z SRGS/ABNF se tento element převede na část grafu v OpenFST. Pokud je identifikován element, který převodník ignoruje, tak se jde dál. Analyzátor tedy dokáže zpracovat celý standart SRGS/ABNF aniž by skončil chybou.



Obrázek 3.8: Vizualizace převodníku G (zpracované gramatiky) v OpenFST.

Po převedení gramatiky na konečný automat se k automatu přidá stav s označením 0, který má orientovanou hranu vedoucí z prvního stavu v automatu. Tato hrana má na svém vstupu i výstupu symbol `<s>`. Stejně tak je přidán koncový stav s orientovanou hranou vedoucí do posledního stavu automatu. Tato nová hrana má za svůj vstupní i výstupní symbol `</s>`. Tyto symboly označují začátek a konec rozpoznané sekvence slov.

Po vytvoření VKA jsou aplikovány operace determinizace a minimalizace, aby se zmenšila jeho velikost a vznikl VKA bez ε přechodů.

3.4 ANTRL

Je nástroj napsaný v Javě pro generování kódu syntaktické a lexikální analýzy nadefinovaného domenového[2] jazyka. Kódy mohou být kromě Javy generovány do jazyků jako Python a C. Konkrétně generátoru pro jazyk C bylo využito v tomto projektu.

Výhodou ANTRL je menší množství kódu které musí programátor pro zpracování vstupu vytvořit. A důsledná kontrola vstupu, zda odpovídá nadefinovanému schématu.

[2]Takto je označován jazyk pro specifické použití.

Další vlastností, kterou ale implementace pro BSAPI nepoužívá, je tvorba abstraktního syntaktického stromu.

3.4.1 ANTRL pravidla pro SGRS/ABNF

Se dělí na lexikální a sémantická pravidla. Jména lexikálních pravidel začínají velkými písmeny a jména syntaktických pravidel zase malými. Za jménem následuje znak : za níž je definice pravidla. Řetězce v apostrofech jsou definicí lexému.

Lexikální analýza očekává vstup ve formátu utf-8. SRGS určuje že se pro definici číslic a znaků mají použít stejné symboly jako v XML[3]. Ukázka několika dalších definovaných lexikálních pravidel:

```
WS :  (' '|'\r'|'\t'|'\u000C'|'\n') {$channel=HIDDEN;} ;

fragment Letter :  BaseChar | Ideographic;

fragment NameChar:
    Letter | Digit | '.' | '-' | '_' | ':' | CombiningChar |
       Extender
    ;
fragment Name:
   (Letter | '_' | ':') (NameChar)*
   ;
fragment Names:
     Name (WS Name)*
     ;
fragment Nmtoken:
    (NameChar)+
    ;

fragment Nmtokens:
        Nmtoken (WS Nmtoken)*
        ;

DoubleQuotedCharacters :
    '"' ('\u0000'..'\u0021'|'\u0023'..'\uFFFE')* '"'
    ;

Token:
    (Letter)+
  | DoubleQuotedCharacters
  ;

RuleName:
    '$' Name
    ;

ML_COMMENT : '/*' ( options {greedy=false;} : . )* '*/' {$channel=
   HIDDEN;};
SL_COMMENT : '//' ( options {greedy=false;} : . )* '\n' {$channel=
   HIDDEN;};
```

[3]Seznam povolených symbolů na webu http://www.w3.org/TR/2000/REC-xml-20001006#NT-Letter

Klíčove slovo `fragment` jazyka ANTRL říká, že definované lexikální pravidlo netvoří lexémy, ale dá se pouze použít jako část dalšího lexikálního pravidla.

Sémantická pravidla pak kombinují lexikální pravidla, lexémy a jiná sémantické pravidla. Základní sémantická pravidla pro jazyk SRGS/ABNF v jazyku ANTRL:

```
mainRule:
 declaration* (( ruleDefinition | sequence  )  )* EOF
;

declaration : rootRuleDecl;
rootRuleDecl: 'root' RuleName ';' ;

ruleDefinition: ruleScope? ruleName=RuleName '=' ruleExpansion ';' ;

ruleScope: 'private' | 'public';

ruleExpansion : ruleAlternative ( '|' ruleAlternative )* ;

sequence: (sequenceElement)+ ';' ;

ruleAlternative: (sequenceElement)+ ;

blockExpansion : blockAlternative ( '|' blockAlternative )* ;

blockAlternative: (sequenceElement)+ ;

sequenceElement: subexpansion (repeatOperator)? ;

subexpansion:
     readToken
   | ruleRef
   | tag
   | '(' choiceOne ')' languageAttachment?
   | '[' optionalOne ']' languageAttachment?
        /* 0 or 1 */
  ;

readToken:
  Token
  languageAttachment?;

choiceOne: blockExpansion;

optionalOne: blockExpansion;

ruleRef: localRuleRef | specialRuleRef | externalRuleRef;

localRuleRef: RuleName ;

specialRuleRef: '$NULL' | '$VOID' | '$GARBAGE';

externalRuleRef: '$' '<'Token '>' ('~''<' Token'>')? ;
```

```
tag: tagPos=('{' .* '}' | '{!{' .* '}!}' ) ;

repeatOperator:
  '<' Number ( ('-')? ( Number )? )'>' ;

languageAttachment: '!' .* ';' ;
```

Vstupem pro sémantickou analýzu gramatiky je pravidlo `mainRule`, které v implementaci vrací seznam vygenerovaných stavů a hran mezi nimi. Podle vrácených dat se sestaví síť převodníku G za pomocí knihovny OpenFst.

3.4.2 Zpracování chyb

Pokud je při analýze objevena chyby ve vstupu je zavolána předdefinována metoda pro její výpis uživateli. Na standardní chybový výstup vytiskne dosti nesrozumitelné hlášení. Toto chování se dá obejít náhradou této funkce.

Program pro analýzu SRGS/ABNF gramatik chybový callbak přepisuje funkcí `myDisplayRecognitionError`. Vstupem je struktura[4] `ANTLR3_BASE_RECOGNIZER`, která nese informace o současném stavu průběhu analýzy. Funkce chybu zpracuje a uloží, aby mohla být předána do BSAPI. Její implementace je zde:

```
void myDisplayRecognitionError
 (pANTLR3_BASE_RECOGNIZER grammarAnalyser,
  pANTLR3_UINT8 * tokenNames)
{
                //grammarAnalyser->state->exception->token
  long line     = grammarAnalyser->state->exception->line;
  long position = grammarAnalyser->state->exception->
     charPositionInLine;

  char * errorText = (char*)grammarAnalyser->state->exception->name;

  switch(recognizer->state->exception->type)
  {
    case ANTLR3_RECOGNITION_EXCEPTION:
         break;
    case ANTLR3_MISMATCHED_TOKEN_EXCEPTION:
         break;
    case ANTLR3_NO_VIABLE_ALT_EXCEPTION:
         break;
    case ANTLR3_MISMATCHED_SET_EXCEPTION:
         break;
    case ANTLR3_EARLY_EXIT_EXCEPTION:
         break;
    case ANTLR3_FAILED_PREDICATE_EXCEPTION:
         break;
    default:
        break;
    }
}
```

[4] Dokumentace k struktuře reprezentující chybu během analýzy je na webu
http://antlr3.org/api/C/struct_a_n_t_l_r3___e_x_c_e_p_t_i_o_n_d_struct.html

V definici funkce je vidět rozdělení chyb na několik typů. Kde třeba typ `ANTLR3_MISMATCHED_TOKEN_EXCEPTION` znamená, že další načtený lexém neodpovídá žádné možnosti, která je definována pravidlem.

Uživatelsky nejpříjemnější metodou zpracování chyb by bylo potlačení callbacu (odkaz na funkci) a analyzování abstraktního syntaktického stromu, který usnadňuje zotavení z chybového stavu a pokračování v analýze.

Kapitola 4

Sestavení rozpoznávací sítě

Jádrem sestavení rozpoznávací sítě je vzorec $HCLG = H \circ min\left(C \circ det\left(L \circ G\right)\right)$ vyplývající z článku [7]. Pro praktickou aplikaci s knihovnou OpenFst, je potřeba vzorec rozšířit na:

$$LG = arcsort_{olabel}\left(rmepsilon\left(L\right)\right) \circ arcsort_{ilabel}\left(rmepsilon\left(G\right)\right) \tag{4.1}$$

$$CLG = arcsort_{olabel}\left(rmepsilon\left(C\right)\right) \circ arcsort_{ilabel}\left(det\left(rmepsilon\left(LG\right)\right)\right) \tag{4.2}$$

$$HCLG = arcsort_{olabel}\left(H\right) \circ arcsort_{ilabel}\left(min\left(det\left(rmepsilon\left(CLG\right)\right)\right)\right) \tag{4.3}$$

Tento vzorec pak vytvoří výstup v podobě převodníku, kde každá hrana má svůj vstupní symbol, výstupní symbol, a pravděpodobnost. Situaci zachycuje výřez $HCLG$ převodníku vytvořeného v OpenFst:

```
...
3   4   sil_4   eps 0.0651507005
4   2   sil_2   eps 0.105728
4   4   sil_4   eps 0.00762159005
4   5   eps   eps 0.0566494986
5   6   z_s2_2   ZERO 2.30273438
5   7   s_s2_1   SEVEN 2.30273438
...
```

První dva sloupce jsou ID patřící uzlům v grafu. První je uzel, ze kterého vede hrana, a druhý je uzel, do kterého vede hrana. Třetí sloupec je akustická jednotka čtvrtý sloupec je slovo. A jako poslední je uložena pravděpodobnost přechodu přes hranu.

4.1 Format rozpoznávací sítě v BSAPI

Který dekodér načítá je unikátní pro projekt. Od převodníku v OpenFst se liší tím, že má symbol z hrany převeden do uzlu. Proto se při sestavení rozpoznávací sítě v BSAPI provádí několik kroku navíc:

- Převod na akceptor $HCLGa$ se slovními a akustickými uzly.
- Převod na formát pro BSAPI.
- Setřídění grafu podle počátečního uzlu a přechod na numerické číslovaní uzlu.

Tyto kroky umožňují snadný zásah a úpravu výsledné rozpoznávací sítě. Pro případ doplnění speciální funkcionality dekodéru, která mu může být předána skrze síť.

BSAPI podporuje binární i textový formát sítě. Textový dovoluje snadnější vývoj a kontrolu zásahu do sítě. Je podporován i z historických důvodů, protože ho BSAPI používalo před binárním a proto že dosavadní metody, tvoření sítí, byly postaveny nad linuxovými skripty. Výhodou binárního formátu sítě je zase rychlost, se kterou ji dekodér načítás.

4.1.1 Převod na akceptor *HCLGa*

VKA je převodník s totožným vstupním i výstupním symbolem. Aby dekodér, který bude používat rozpoznávací síť rozeznal, který symbol je akustický model a který je slovo jsou symboly rozšířené o předponu "`W=`"nebo "`M=`". "`W=`"jako "word"je předpona pro symboly z G a "`M=`"jako model je předpona pro symboly z H. Například:

```
0       1       W=<s>   W=<s>
1       2       M=sil_2 M=sil_2 -2.30258512
2       2       M=sil_2 M=sil_2 0.00865190011
...
```

Druhý a třetí sloupec pro vstupní a výstupní symboly obsahuje stejné hodnoty. Toho lze docílit algoritmem 3. Kde:

- **.** lze představit jako operaci konkatenace dvou řetězců.
- && je symbolem pro logické and, stejně jako v podmínce jazyka C++.

Data: Převodník $HCLG = (A, B, Q_T, I, F_T, E, \lambda, p_T)$ s jedním počátečním stavem.
Result: Akceptor $HCLGa = (Q_A, \sum, \sigma, s, F_A, p_A)$
method
$s \leftarrow I$
$F_A \leftarrow F_T$
$p_A \leftarrow p_T$
$Q_A \leftarrow Q_T$
while $Q_T \neq \emptyset$ **do**
  $state \leftarrow HEAD(Q_T)$
  $DEQUEUE(Q_T)$
  for *each* $h, h \in E$ && $h \in (state, input_l, output_l, likelihood, next_s)$ **do**
    $\sum \leftarrow \sum \cup$"$M =$ " **.** $input_l$,"$W =$ " **.** $output_l$ **if** $input_l = \varepsilon$ && $output_l = \varepsilon$ **then**
      $\sigma \leftarrow \sigma \cup (state, \varepsilon, likelihood, next_s)$
    end
    if $input_l = \varepsilon$ && $output_l \neq \varepsilon$ **then**
      $\sigma \leftarrow \sigma \cup (state,$"$W =$ " **.** $output_l, likelihood, next_s)$
    end
    if $input_l \neq \varepsilon$ && $output_l = \varepsilon$ **then**
      $\sigma \leftarrow \sigma \cup (state,$"$M =$ " **.** $input_l, likelihood, next_s)$
    end
    if $input_l \neq \varepsilon$ && $output_l \neq \varepsilon$ **then**
      $Q_A \leftarrow \cup new_s$
      $\sigma \leftarrow \sigma \cup (state,$"$M =$ " **.** $input_l, likelihood, new_s)$
      $\sigma \leftarrow \sigma \cup (new_s,$"$W =$ " **.** $output_l, 0, next_s)$
    end
  end
end

Algorithm 3: Metoda převodu převodníku T na akceptor A.

Dalšími kroky v převodu je determinizace podle [7] , kterou lze na $HCLGa$ provést díky algoritmu 1 . Pak následuje přemapování pomocných rozlišovacích symbolů na ε. Pomocné symboly začínající znakem **#**, které slouží k rozlišení stejně znějících slov v převodníku L, tak aby šel determinizovat. Tedy můžeme zapsat:
$determinize\,(map\,(\#?, \varepsilon))$

4.1.2 Převod na formát pro BSAPI

Převodem na mezisíť se usnadní převod na rozpoznávací síť. Formát mezi sítě je zachycen zde:

```
n0 0 l=-0
0 W=<s> n1
n1 1 l=-0
1 M=sil_2 n2
n2 2 l=-0.0579227
2 M=sil_4 n4
n2 3 l=-0.0861414
3 M=sil_3 n3
n2 1 l=-0.0086519
n3 2 l=-0.0651507
...
```

```
n422 n425 l=-0.0566495
...
```

Jak je vidět formát BSAPI obsahuje 3 typy záznamu. První je spojení mezi uzly `n1 1 l=-0`, druhý je spojení se symbolem `0 W=<s> n1` a třetí je spojení ke koncovému uzlu `n422 n425 l=-0.0566495`. Lépe to vysvětluje následující výřez z poznámek ve zdrojových kódech pro uchování tohoto formátu:

```
* Which describe node with label and connetion
* to other nodes with path likelihood.
*  One record (describe or connection) is on one line.
*   node with label syntax example:
*     "0 W=<s> n1"
*      |   |   '- mToNode              = 1, is pointer to node1
*      |   '----- mNodeLabel           = "W=<s>"
*      '--------- mFromNode (node id ) = 0
*                 mLikelihood          = 0.0   (implicit)
*
*  and connection to other node
*     "n2 2 l=-0.0687369"
*      |  | '------------- likelihood = -0.0687369
*      |  '--------------- toNode     = 2
*      '------------------ fromNode   = 2
*                         nodeLabel  = "" (implicit)
*
*  and connection to other node (end node)
*     "n194 n195 l=-0.0614271"
*      |     |    '----------- likelihood = -0.0614271
*      |     '---------------- toNode     = 195
*      '---------------------- fromNode   = 194
*                              nodeLabel  = "" (implicit)
```

Takovéhoto formátu lze docílit algoritmem 4, kde funkce $NMAP$ mapuje string na unikátní ID a pro stejný string vrátí stejné ID. Pro jeho potřeby si mezisiť TNET zadefinujme jako trojici nad okruhem K s reálnými čísly, který je použit pro vyjádření pravděpodobnosti:
$TNET = (Q, \sum, \delta)$, kde:
Q je konečná množina stavů.
$\sum$ je konečná množina symbolů.
δ je množina přechodů,
taková že $\delta \subseteq Q \times \sum \times K \times type \in \{Label, Connection, ConnectionToEnd\} \times Q$ kde první stav značí startovní symbol přechodu a druhý stav značí koncový stav přechodu.

Data: Deterministický akceptor $HCLGa = (Q_A, \sum_A, \sigma, s, F, p)$, bez ε symbolů.
Result: $TNET = (Q, \sum, \delta)$
Method:
$\sum \leftarrow \sum_A$
$\delta \leftarrow \{\varepsilon\}$
$Q \leftarrow Q_A$
forall the $(from_s, symbol, likelihood, to_s) \in \sigma$ **do**
  if $symbol = \varepsilon$ **then**
    $\delta \leftarrow \delta \cup \{(from_s, \varepsilon, 0 - likelihood, Connection, to_s)\}$
  end
  if $symbol \neq \varepsilon$ **then**
    $m \leftarrow symbol \,.\text{"\_"}\, .ToString\,(to_s)$
    $Q \leftarrow Q \cup NMAP\,\{(m)\}$
    $\delta \leftarrow \delta \cup \{(from_s, \varepsilon, 0 - likelihood, ConnectionToEnd, NMAP\,(m))\}$
    if $to_s \notin S$ **then**
      $S \leftarrow S \cup \{to_s\}$
      $\delta \leftarrow \delta \cup \{(NMAP\,(m)\,, symbol, 0, Label, to_s)\}$
    end
  end
end

Algorithm 4: Převod na formát pro BSAPI.

4.1.3 Optimalizace BSAPI sítě

Obnáší setřídění podle počátečního uzlu a numerické setřídění uzlů.

Formát textového souboru s optimalizovanou BSAPI sítí:

```
N=467 L=946
0 W=!NULL 1 l=0
1 W=<s> 2 l=0
2 M=sil_2 3 l=-0.05792 4 l=-0.08614 2 l=-0.00865
3 M=sil_4 5 l=-2.35938 6 l=-2.35938 7 l=-1.66623
8 l=-2.35938 9 l=-2.35938 10 l=-2.35938 11 l=-1.66623
12 l=-2.35938 3 l=-0.00762 2 l=-0.10572
...
```

N na prvním řádků udává počet uzlů v síti a L počet hran v síti.

Pro potřeby algoritmu 5, který setřídí síť zadefinujme objekt NET nad okruhem K. Okruh slouží pro vyjádření pravděpodobností přechodů ze stavu do stavu:
$NET = (Q, \sum, \varrho, \phi)$,kde:
Q je konečná množina stavů sítě.
$\sum$ je konečná množina symbolů.
ϱ je funkce mapující symbol na stav $\varrho : Q \to \sum$
ϕ je množina přechodů ze stavu do stavu, taková že $\phi \subseteq Q \times K \times Q$

Data: $TNET = (Q_T, \sum_T, \delta)$
Result: $NET = (Q, \sum, \varrho, \phi)$
Method:
while $Q_T \neq \emptyset$ **do**
  $n \leftarrow HEAD\,(Q_T)$
  $DEQUEUE\,(Q_T)$
  $Q \leftarrow Q \cup \{n\}$
  for *each* $(from_s, symbol, likelihood, ConnectionType, to_s) \in \delta$ **do**
    if $from_s = n$ *and* $ConnectionType = Label$ **then**
      $\varrho \leftarrow \{(x, \_) \in \varrho\}$, where $x \neq n$
      $\varrho \leftarrow \varrho \cup \{\{(n, symbol)\}\}$
    end
    if $from_s = n$ *and* $ConnectionType \neq Label$ **then**
      $\phi \leftarrow \phi \cup \{(n, likelihood, to_s)\}$
    end
  end
end

Algorithm 5: Optimalizace BSAPI sítě.

4.1.4 Vytvoření převodníku L

Převodník L se tvoří každé gramatice na míru. Protože převodník obsahující všechna slova v jazyku by byl příliš velký a s takovým převodníkem by se během sestavování rozpoznávací sítě pracovalo pomalu. Navíc všechna slova v daném jazyku ani obsahovat nemůže.

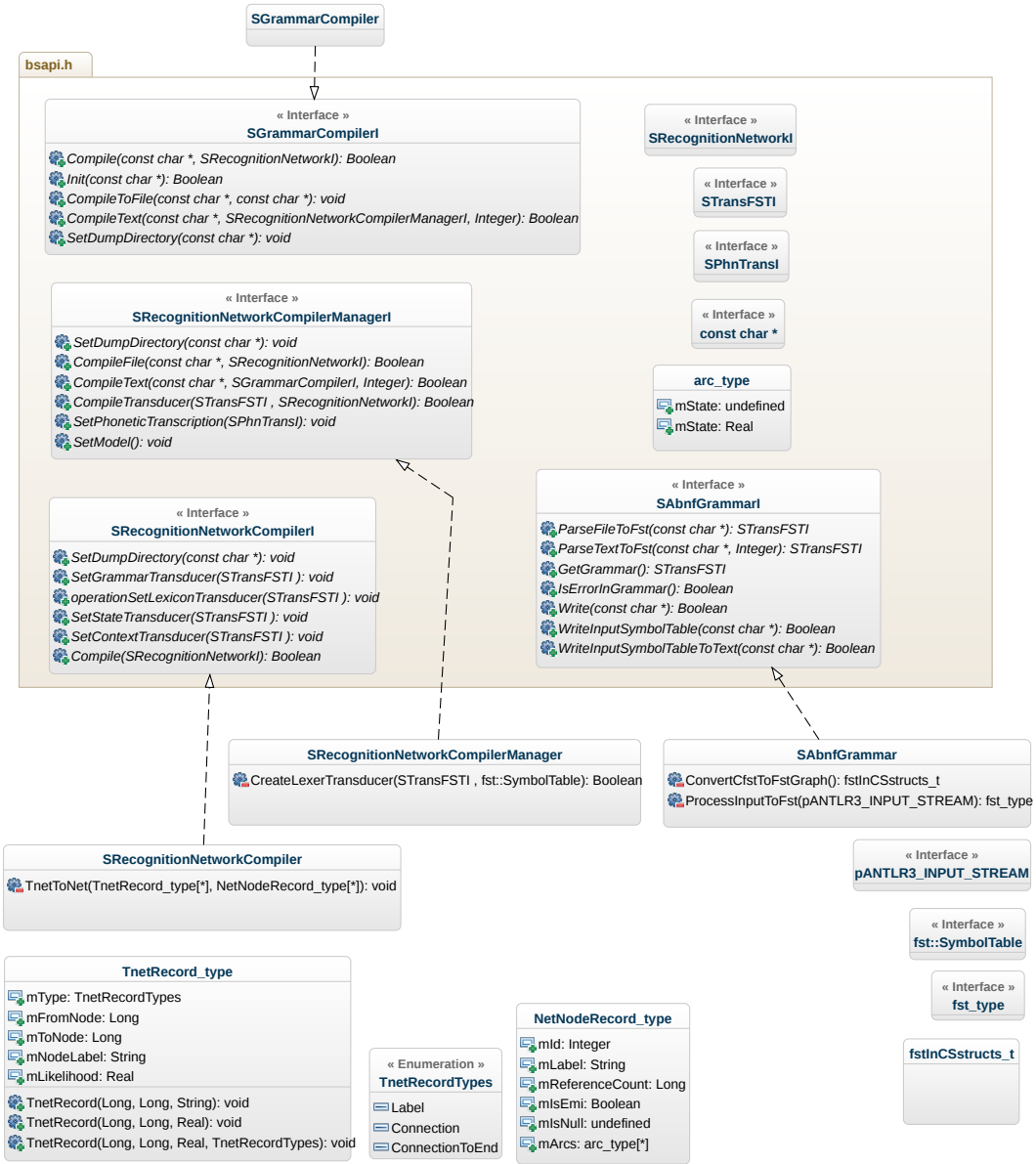
V BSAPI slouží pro překlad slov na fonémy rozhraní SPhnTrasI, které vrací fonémový přepis slova. To se nejdříve pokusí vyhledat slovo ve slovníků známých výslovností a pokud ho nenajde tak vrátí přepis grafému na fonémy. Grafém je znak z abecedy. Přepis je definován pro každý jazyk. Přepis grafémů vrací nesprávnou výslovnost především pro slova, která jazyk přejal z jiného jazyku a jenž vyslovuje variantou originální výslovnosti.

4.2 Zapouzdření do BSAPI

Obrázek 4.2 je UML (Unified Modeling Language) diagramem tříd zachycující objekty zpracovávající SRGS/ABNF gramatiku a objekty pro sestavení rozpoznávací sítě. V diagramu je znázorněn soubor bsapi.h, ve kterém jsou definovány všechny rozhraní, které knihovna používá pro komunikaci. Také to je hlavičkový soubor, který se dodává zákazníkům. Jednotlivé objekty blíže vysvětluje tabulka 4.2.

Jméno objektu	**Popis objektu**
SGramarCompilerI	rozhraní kompilátoru sítě.
SGramarCompiler	Implementace rozhraní SGramarCompilerI.
SRecognitionNetworkI	Rozhraní pro práci s rozpoznávací sítí.
SRecognitionNetwork	Implementace rozhraní SRecognitionNetworkI.
STransFSTI	Rozhraní pro práci s konečnými automaty.
SPhnTrasI	Rozhraní pro přepis řetězce reprezentující slovo na jeho fonémy. Používá se při sestavování převodníku L.
arc_type	Struktura popisující hranu sítě, Obsahující cílový stav a pravděpodobnost hrany.
SRecognitionNetworkCompilerManagerI	Rozhraní pro přípravu převodníků pro sestavení rozpoznávací sítě.
SRecognitionNetworkCompilerManager	Implementace rozhraní SRecognitionNetworkCompilerManagerI.
SAbnfGrammarI	Rozhraní pro načítaní a zpracování SRGS/ABNF gramatiky
SAbnfGrammar	Implementace rozhraní SAbnfGrammarI.
SRecognitionNetworkCompilerI	Rozhraní pro sestavení rozpoznávací sítě dle kapitoly 4
SRecognitionNetworkCompiler	Implementace rozhraní SRecognitionNetworkCompilerI

Tabulka 4.1: Zodpovědnosti jednotlivých objektů.



Obrázek 4.1: Zapozdření tříd pro sestavení rozpoznávací sítě do BSAPI.

Kapitola 5

Součinnost gramatik s rozpoznávačem

Funguje skrze rozpoznávací sítě, které jsou vytvořeny podle gramatiky a poté načteny dekodérem rozpoznávače. Rozpoznávač poté hledá nelepší průchod rozpoznávací sítí, tak aby odpovídal zpracovávané řeči.

5.1 Testy sestavení rozpoznávací sítě

Jsem prováděl, protože mnou sestavené rozpoznávací sítě se liší od původní sítě vytvořené skriptem. Při vytváření sítí jsem totiž neimplementoval poslední krok, kterým je optimalizace sítě nástrojem `SExpand` z projektu [1].

Testy jsem prováděl na stroji s procesorem i5-3470[2].

Přesnost, rychlost a paměťové nároky rozpoznávací sítě jsem testoval na sítích pro přepis řeči, kde gramatika dovoluje všechny kombinace slov. Více v o statistickém popisu jazykových modelů v článku [1] a [8]. Tato úprava mi dovolí využít připravené testy nad sadami nahrávek pro testování jazykových modelů. Díky tomu jsem dostal objektivnější výsledky, protože jsem je mohl srovnat s předešlou metodou tvoření rozpoznávací sítě. Testy byly provedeny na testovacích sadách pro CZ (český jazyk) a RU (ruský jazyk).

5.1.1 Rychlost načtení gramatiky a sestavení rozpoznávací sítě

Byla testována na operačním systému Linux s pomocí knihovny `<sys/time.h>` z prostředí jazyka `C++`. Před a za kód pro zpracování vstupu a získání rozpoznávací sítě jsem vložil kód pro zaznamenání času.

Doba zpracování vstupu je závislá na jeho velikosti, počtu pravidel a na jejich vzájemném propojení.

Ukázková gramatika pro objednání pizzy 3.1 se zpracovávala do rozpoznávací sítě zhruba za 0,158 s. Tento čas je výrazně kratší než 3 minuty, které byly zapotřebí pro sestavení souboru obsahující ekvivalentní rozpoznávací síť starou metodou vytvořenou ve skriptech.

Do měřeného času není započítán čas pro inicializaci objektu pro `SRecognitionNetworkCompilerManager`, která se provádí jen jednou. Po inicializaci se

[1] více o STK na webu http://speech.fit.vutbr.cz/cs/software/hmm-toolkit-stk

[2] Procesor Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz, více na webu http://ark.intel.com/products/68316/Intel-Core-i5-3470-Processor-6M-Cache-up-to-3_60-GHz

Síť	WER [%]	WA [%]	WC [%]
CZ ze skriptu	35,2	67,2	64,8
CZ implementace	35,2	67,2	64,8
RU ze skriptu	52,6	51,5	47.4
RU implementace	52,6	51,5	47.4

Tabulka 5.1: Přesností zpracování.

může sestavení rozpoznávací sítě z gramatiky libovolně krát opakovat. U staré metody bylo zapotřebí po každém spuštění zapotřebí znovu načíst vstupní modely.

5.1.2 Přesnost rozpoznávací sítě

Provádím, abych zjistil jak vynechání optimalizace rozpoznávací sítě nástrojem `SExpand` ovlivnilo přesnost rozpoznávání s touto sítí. Pro vyhodnocení přesnosti rozpoznávání řeči se používají testovací sady nahrávek s referenčním přepis. Ve výsledcích testovaného rozpoznávače řeči, výsledkem je její přepis, se pro každou nahrávku sledují tyto parametry:

R (References) počet referenčních slov.

C (Correct) počet správně rozpoznaných slov na pozici referenčních slov.

S (Substitution) počet špatně rozpoznaných slov.

I (Insertions) počet vložených slov, mimo místa s referenčními slovy.

D (Deletions) počet míst, kde je referenční slovo, ale rozpoznávač nic nerozpoznal.

Podle článků [9] se hlídají ukazatele jako WER (Word Error Rate), WA (Word Accuracy), WC (Word Correctness). Které se vypočítají následovně:

$$WC = \frac{C}{R} \times 100 \tag{5.1}$$

$$WER = \frac{S + D + I}{R} \times 100 \tag{5.2}$$

$$WA = 1 - WER \tag{5.3}$$

Z tabulky 5.1 je vidět, že implementace skládání rozpoznávacích sítí je generuje ekvivalentní výstupy k předešlému vytváření sítí pomocí skriptů.

5.1.3 Rychlost rozpoznávací sítě

Provádím, abych zjistil jak vynechání optimalizace rozpoznávací sítě nástrojem `SExpand` ovlivnilo rychlost rozpoznávání s touto sítí.

Z tabulky 5.2 pro češtinu je poměr součtu dob zpracování nahrávek k součtu dob trvání nahrávek pak dává u původní metody tvorby rozpoznávacích sítí hodnotu 0,41 a u současné implementace v BSAPI pak 0,43.

Z tabulky 5.3 pro ruštinu je poměr součtu dob zpracování nahrávek k součtu dob trvání nahrávek pak dává u původní metody tvorby rozpoznávacích sítí hodnotu 0,48 a u současné implementace v BSAPI pak 0,49.

Nahrávka	Délka [s]	Původní [s]	Nový [s]
wav_prom060905_0003.wav	20,92	8,87	9,15
wav_prom060907_0006.wav	80,39	35,67	37,64
wav_prom060907_029a.wav	31,16	13,17	13,39
wav_prom060907_032a.wav	42,95	17,24	17,63
wav_prom060907_111a.wav	75,21	29,21	30,51

Tabulka 5.2: Časy zpracování nahrávek pro CZ jazykový model.

Nahrávka	Délka [s]	Původní [s]	Nový [s]
wav_ru3467.wav	74,34	36.71	37,60
wav_ru6535.wav	72,92	28,34	28,94
wav_ru6723b.wav	136,00	61,73	63,19
wav_ru7028.wav	61,91	32,90	33,84
wav_ru7914.wav	65,52	23,34	23,98
wav_ruN0001.wav	30,40	18,22	18,52
wav_ruN0002.wav	51,77	35,09	35,87

Tabulka 5.3: Časy zpracování nahrávek pro RU jazykový model.

Tato metoda měření není přesná, protože nezohledňuje množství řeči v nahrávce, která je zpracovávána. Ale přesto nabízí srovnání obou postupů a ukazuje, že současná implementace je pomalejší. Nárůst času potřebný pro zpracování nahrávky je sice malý, ale ne nezanedbatelný, protože se projeví při práci s velkým množství dat.

5.1.4 Paměťové nároky rozpoznávací sítě

Provádím, abych zjistil jak vynechání optimalizace rozpoznávací sítě nástrojem `SExpand` ovlivnilo množství paměti potřebné pro rozpoznávací síť. Jsem sledoval linuxovým nástrojem `top`, který zobrazuje množství operační paměti, kterou konzumuje proces. Výsledky jsou zachyceny v tabulce 5.4, kde je zaznamená velikost velikost souborů pro ekvivalentní rozpoznávací sítě. Jedna vyrobená původním skriptem a druhá vytvořená kompilátorem.

Dále je v tabulce 5.4 zaznamenána největší vytíženost operační paměti RAM (Rndom Access Memory), kterou zabírá proces rozpoznávače. Velikost obsazené paměti rozpoznávačem je ovlivněna především velikostí sítě a velikostí nahrávky. Proto se tento údaj musí chápat jako srovnávací, protože jediným rozdílem v podmínkách testů byla vstupní síť s jazykovým modelem pro rozpoznávání.

Z tabulky 5.4 je vidět že nová implementace sestavení rozpoznávacích sítí generuje větší paměťové struktury.

Jazyk	Velikost původní sítě	Velikost nové sítě	RAM původní	RAM nové
CZ	290 MB	488 MB	26% z 8 GB	34% z 8 GB
RU	299 MB	458 MB	17,6% z 8 GB	24,5% z 8GB

Tabulka 5.4: Vytíženost RAM.

5.1.5 Zhodnocení testů

Z testů rozpoznávání je vidět že implementovaný postup generuje rozpoznávací sítě, které jsou větší, pomalejší, a práce s nimi zabírá více operační paměti, ale při jejích použití se dosahuje stejných přesností jako u sítí, které byli vygenerovány skriptem. Důvodem je že implementovaná metoda vynechává poslední krok ze skriptu, kterým je nástroj `SExpand` z projektu STK[3]. Ten implementuje další optimalizační metody.

[3] více o STK na webu http://speech.fit.vutbr.cz/cs/software/hmm-toolkit-stk

Kapitola 6

Rozpoznávač jako modul UniMRCP

K implementaci modulu pro UniMRCP byl použit dekodér pro rozpoznávače pracujícím s aktuálním časem, který pro rozpoznání a přepis textu používá pouze část zvukové stopy od jejího začátku po aktuálně zpracovávaný rámec řeči. Jednoduše si to lze představit jako rozpoznávání řeči pronášené do mikrofonu. Tento postup odpovídá zpracováváním krátkých promluv, kde se rozpoznávač musí sám rozhodnout zda rozpoznávání ukončit, či dále pracovat.

6.1 Plugin UniMRCP

Plugin je definovaný rozhraním, kterým je struktura jazyka C obsahující ukazatele na funkce[1] a jiné struktury. Rozhraní je definováno pro každý mediální zdroj, tedy rozpoznávání, nahrávání, verifikaci a převod textu na řeč.

UniMRCP obsahuje server a řízení jeho stavu na základě zpráv protokolu MRCP, takže toto nemusí programátor pluginu řešit. Ten se tak může více soustředit na implementaci. Implementovanému se v terminologii UniMRCP říká "engine".

6.1.1 Vytvoření nového pluginu

UniMRCP je postaveno na nástroji `autoconfig`[2], proto je nutné přidat informace o novém pluginu[3] do konfiguračních souboru v projektu.

6.1.2 Konfigurace pluginu

V souboru `unimrcp/conf/unimrcpserver.xml` je v XML tagu `engine` hlavní konfigurace pluginu. Kde je uloženo jméno enginu, podle něhož se hledá dynamická knihovna s jeho implementací. Tato knihovna se načte a začne pracovat, pokud je parametr `enable` nastaven na `true`. Konfigurační záznam pluginu rozpoznávače pro Phonexii:

[1] Ukazatel na funkci se v terminologii jazyka C také někdy nazývá callback.

[2] Nástroj pro generovaní make skriptů. Více na webu: http://www.gnu.org/software/autoconf/

[3] Podrobnější popis na webu:
http://www.wikihouse.com/pman10/index.php?Linux%A5%E1%A5%E2%2FUniMRCP%2F%A5%D7%A5%E9%A5%B0%A5%A4%A5%F3%C4%C9%B2%C3%CA%FD%CB%A1

```
<engine id="Phonexia-ASR" name="phonexiaasr" enable="true">
  <max-channel-count>1</max-channel-count>
  <!-- . is folder where is this xml config file  -->
  <param name="bsapi-license-file"     value=""/>
  <param name="bsapi-setting-file"     value="./../../../../
     Phonexia-ASR-data/settings/config_onlinerec_czech"/>
  <param name="bsapi-gc-setting-file"  value="./../../../../
     Phonexia-ASR-data/settings/config_gc_czech"/>
</engine>
```

V tagu `max-channel-count` je uložen maximální počet VOIP telefonátu, které může engin obsluhovat v jeden okamžik. Další parametry pak definují konfigurační proměnné, které jsou skrze rozhraní pluginu dostupné zevnitř enginu.

Přes parametr `bsapi-gc-setting-file` se BSAPI uvnitř pluginu dozví jméno konfiguračního souboru pro proces nastavení pro sestavení rozpoznávací sítě z gramatiky.

6.1.3 Rozhraní pluginu rozpoznávače

Obrázek 6.1 zachycuje celé rozhraní, které se dělí na části pro jádro rozpoznávače, jádro modulu a zpracování kanálů.

Jádro modulu

Je uloženo ve struktuře `asr_engine_t`. Ukazatele na funkce, které s ním pracují jsou uloženy v `mrcp_engine_method_vtable_t`. Na začátku je zavolána funkce `mrcp_engine_open` ve které se modul inicializuje. Vytvoří tolik rozpoznávacích jader kolik je nastaveno kanálu v konfiguraci.

Funkce `mrcp_engine_channel_create` je volaná, pokaždé když se IVR pokouší o vytvoření nového kanálů. V této funkci je ke kanálu přiřazeno rozpoznávací jádro, které ho obsluhuje.

Jádro rozpoznávače

Je uloženo ve struktuře `phonexia_asr_t`. Obsahuje překladač gramatik pro sestavení rozpoznávací sítě a rozpoznávač řeči.

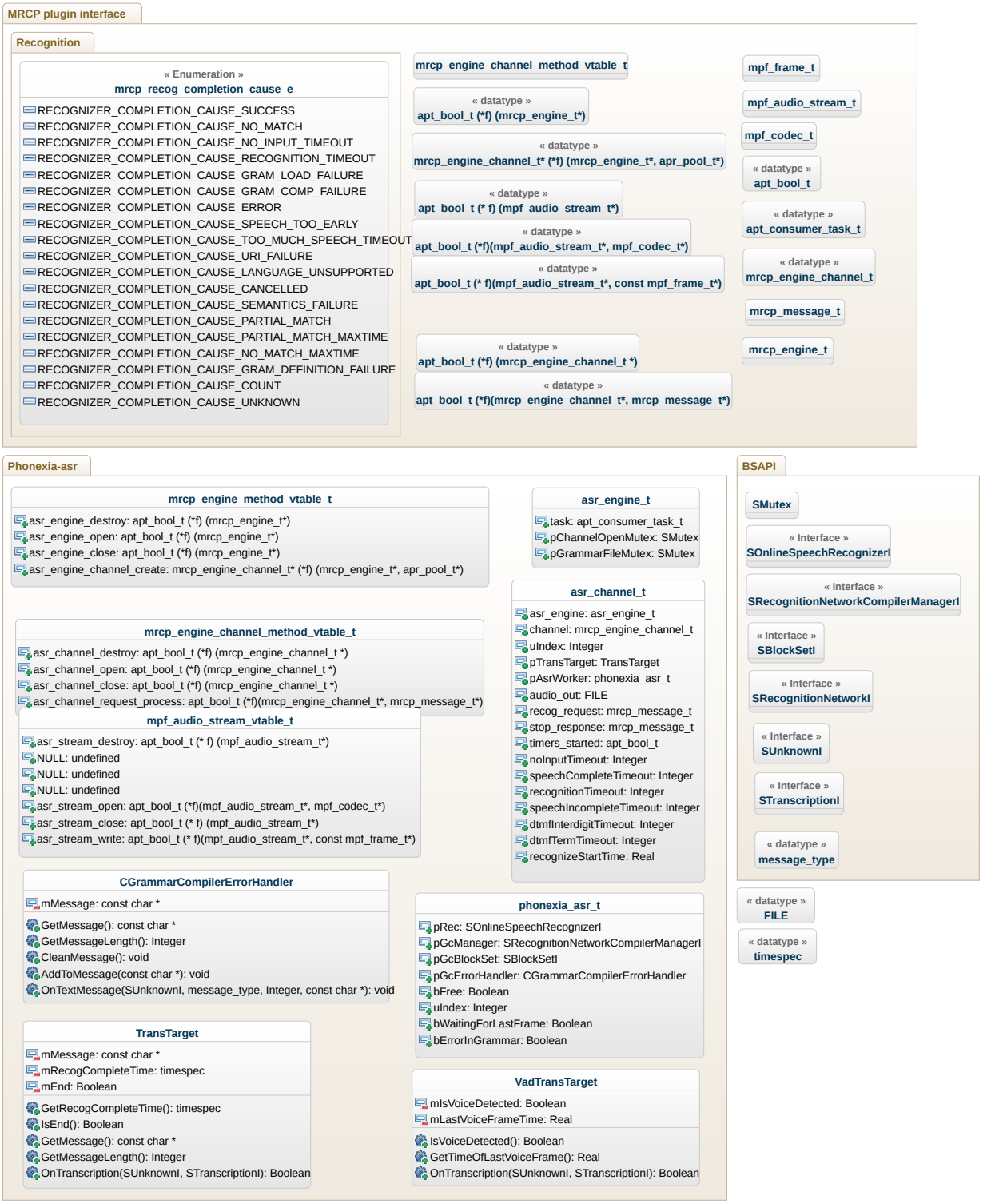
Zpracování kanálů

Struktura `asr_channel_t` obsluhuje spojení dialogového manažera s UniMRCP pluginem.

6.1.4 Ukončení rozpoznávání

Výčtový typ `mrcp_recog_completion_cause_e` obsahuje definované důvody, proč bylo ukončeno rozpoznávání, nebo komunikace po kanálů.

Plugin díky kombinací gramatiky a nastavených časových limitů, sám určí kdy je rozpoznávání ukončeno a pošle výsledek dialogovému manažerovi.



Obrázek 6.1: Diagram UniMRCP modulu pro ASR.

Kapitola 7

Použití UniMRCP ve FreeSwitch

O komunikaci UniMRCP a FreeSwitch se stará modul `mod_unimrcp`[1], který je součástí projektu FreeSwitch.

Pro použití rozpoznávače řeči je zapotřebí nakonfigurování spojení s MRCP serverem, vytvoření dialogových scénářů a napojení tohoto scénáře na SIP telefonní číslo. Dialogový scénář je aplikací, kde vzniká největší přidaná hodnota pro uživatele, protože to on definuje jak systém zareaguje na výsledek rozpoznávání řeči.

7.1 Konfigurace spojení s UniMRCP

Se musí definovat v XML souboru ve složce `conf/mrcp_profiles`. Obsah souboru `phonexiamrcp.xml` pro spojení s UniMRCP:

```
<include>
  <profile name="phonexiamrcp" version="2">
    <param name="client-ip"   value="127.0.0.1"/>
    <param name="client-port" value="5090"/>
    <param name="server-ip"   value="127.0.0.1"/>
    <param name="server-port" value="8060"/>
    <param name="sip-transport" value="udp"/>
    <param name="ua-name"     value="FreeSWITCH"/>
    <param name="rtp-ip"       value="127.0.0.1"/>
    <param name="rtp-port-min" value="4000"/>
    <param name="rtp-port-max" value="5000"/>
    <param name="codecs"       value="PCMU␣PCMA␣L16/96/8000"/>
    <recogparams>
    </recogparams>
  </profile>
</include>
```

Klient v konfiguraci znamená MRCP klient, tedy ten kdo využívá služeb MRCP serveru. Tím je FreeSwitch na kterém je tato konfigurace. Jednotlivé parametry popisuje tabulka 7.1.

[1] Více na webu: https://wiki.freeswitch.org/wiki/Mod_unimrcp

[2] IP (Internet Protocol) adresa slouží pro adresování počítače v internetové sítí. Více v RFC791 na webu: http://tools.ietf.org/html/rfc791

Parametr	Význam
client-ip	Ip[2] adresa FreeSwitch serveru.
client-port	Port FreeSwitch serveru pro protokol MRCP.
server-ip	Ip adresa UniMRCP serveru.
server-port	Port UniMRCP serveru pro protokol MRCP.
sip-transport	protokol pro přenos SIPu. UDP[3] nebo TCP[4].
codecs	Seznam formátů pro přenos audia.
rtp-port-min rtp-port-max	Rozsah portů na serveru UniMRCP pro přenos audia.

Tabulka 7.1: Tabulka parametrů pro konfiguraci MRCP spojení.

7.2 Dialogový scénář

Lze v projektu FreeSwitch sestavit v Perlu, jazyku C, Lua skriptu a Javascriptu. Scénáře jsou uloženy ve složce `freeswitch/scripts`. Scénář pro volání ASR definovaný v Lua skriptu vypadá následovně:

```
session:answer()
session:execute("play_and_detect_speech","say:␣detect:unimrcp␣
   {no-input-timeout=5000␣,recognition-timeout=5000␣,start-
   input-timers=true,define-grammar=true}␣test" )

local speech_result = session:getVariable('
   detect_speech_result')
if (result ~= nil) then
   freeswitch.consoleLog("INFO", speech_result .. "\n")
   message:chat_execute("reply", speech_result);
else
   freeswitch.consoleLog("INFO","No␣result!\n")
end
```

Na začátku je příkazem `session:answer()` zvednuto sluchátko v ústředně. Skript spustí pomocí funkce `execute` spustí metodu `play_and_detect_speech`, který pomocí TTS přehraje argument `say`. V tomto volání není vyplněn. A po přehrání požádá server MRCP o rozpoznání řeči podle gramatiky test. Pokud došlo k úspěšnému rozpoznání je v proměnné `detect_speech_result` uložen výsledek vrácený z MRCP, který se načte do proměnné `speech_result`. V opačném případě se nic nenačte a v proměnné je `nil`.

7.3 Přiřazení dialogového scénáře k telefonnímu číslu

Je definováno v XML souboru `freeswitch/conf/dialplan/public.xml` následovně:

```
<extension name="Phonexia␣ASR">
  <condition field="destination_number" expression="^4001$" >
```

[3]Protokol pro přenos dat UDP (User Datagram protokol). Vice v RFC768 na webu: http://tools.ietf.org/html/rfc768.

[4]Protokol pro přenos dat TCP (Transmission Control Protocol). Vice v RFC793 na webu: http://tools.ietf.org/html/rfc793.

```
    <action application="lua" data="asrdevel.lua" />
  </condition>
</extension>
```

Záznam říká, že při volání čísla 4001 se má spustit dialogový scénář v souboru `asrdevel.lua`, který je napsaný v Lua skriptu. FreeSwitch vyhledává scénáře v Lua skriptu ve složce `freeswitch/scripts`.

Kapitola 8

Závěr

Výsledkem této práce je základ IVR systému pro rozpoznávač řeči řízený SRGS/ABNF gramatikou. Typickým použitím tohoto systému je nahrazení odpovědi pomocí DTMF volby použitím lidské řeči. Dalším použitím je realizace hlasového vstupu pro zařízení ovládána hlasem.

Pro realizování tohoto systému jsem vytvořil překladač podmnožiny SRGS/ABNF gramatik na váhovaný konečný automat. Projekt BSAPI jsem doplnil, tak aby podporoval překladač gramatik. Konečný automat je v něm podporován knihovnou OpenFst. Podle existujících skriptů a článků [7] jsem implementoval sestavování rozpoznávacích sítí v C++ v BSAPI. S použitím BSAPI dekodéru jsem vytvořil plugin pro rozpoznávání řeči pro UniMRCP projekt. Plugin komunikuje se softwarovou telefonní ústřednou FreeSwitch pomocí protokolů MRCPv2 a RTP. Ve FreeSwitchi jsem nakonfiguroval spojení s pluginem pro rozpoznávání řeči a definoval jsem dialogový scénář pro prezentaci systému.

V překladači jsem neimplementoval podporu některých elementů jazyka třeba podpora vícejazykových gramatik, ale především elementu "tag", který slouží pro definovaní sémantické interpretace řeči. Překladač ani neakceptuje vstup v XML formátu. O tuto funkcionalitu by měl být v překladač v budoucnu rozšířen. Dále doplnit implementaci pluginu pro rozpoznávání řeči, jelikož plně nepodporuje zcela všechny možnosti protokolu MRCPv2, a to především důvěryhodnost s jakou rozpoznávač text určil. Proto ale bude potřeba o tuto funkci rozšířit dekodér pro online rozpoznávání řečí v BSPI. A v neposlední řadě zahrnout optimalizaci z nástroje `SExpand` z projektu STK do procesu sestavování sítí.

Díky tomuto projektu jsem získal nové zkušenost z oboru zpracování přirozeného jazyka, konfigurace IP telefonie. Prohloubil své znalosti z oblastí teorie konečných automatů a síťových protokolů. Ale především jsem se seznámil s práci ve firemním prostředí, kde jsem mohl sledovat zpětnou vazbu manažerů a zákazníků na mou práci.

Literatura

[1] A. DEROUAULT, B. M.: Natural Language Modeling for Phoneme-to-Text Transcription.
URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04767855

[2] A. Hunt, S. McGlashan: Speech Recognition Grammar Specification Version 1.0.
URL http://www.w3.org/TR/speech-grammar/

[3] D. Burnett, S. S.: Media Resource Control Protocol Version 2 (MRCPv2), 2012.
URL http://tools.ietf.org/html/rfc6787

[4] D. Povey: The Kaldi Speech Recognition Toolkit.
URL http://homepages.inf.ed.ac.uk/aghoshal/pubs/asru11-kaldi.pdf

[5] Freeform Dynamics: 2005 Speech Recognition Customer Satisfaction Survey. Technická zpráva, 2005.
URL http://www.freeformdynamics.com/misc/docs/Nuance_Customer_Sat_ExecSumm.pdf

[6] M. Mohri: Weighted Finite-State Transducer Algorithms.
URL http://www.cs.nyu.edu/~mohri/pub/fla.pdf

[7] M. Mohri, F. Pereira, M. R.: SPEECH RECOGNITION WITH WEIGHTED FINITE-STATE TRANSDUCERS.
URL http://www.cs.nyu.edu/~mohri/pub/hbka.pdf

[8] R. KUHN, R. M.: A Cache-Based Natural Language Model for Speech Recognition.
URL http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=56193

[9] S. J. Young, L. L. C.: Speech recognition evaluation: a review of the U.S. CSR and LVCSR programmes.
URL
http://ac.els-cdn.com/S0885230898901012/1-s2.0-S0885230898901012-main.pdf?_tid=0388f02c-e4ec-11e3-91cc-00000aab0f27&acdnat=1401118987_0c0bb8b00af46f7872d95d660e56f2ac

[10] T. Parr: *The Definitive ANTLR Reference: Building Domain-Specific Languages.* Pragmatic Programmers, Pragmatic Bookshelf, první vydání, Květen 2007, ISBN 0978739256.
URL http://www.amazon.com/Definitive-ANTLR-Reference-Domain-Specific-Programmers/dp/0978739256%3FSubscriptionId%3D13CT5CVB80YFWJEPWS02%26tag%3Dws%26linkCode%3Dxm2%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0978739256

Příloha A

Obsah CD

- Soubor "CTIME.txt".
- Zdrojové kódy pro převod SRGS/ABNF na konečný automat.
- Zdrojové kódy části BSAPI, které byli upraveny v rámci projektu.
  BSAPI nelze spustit bez licence, pro bližší informace kontaktujte support@phonexia.com
- Zdrojové kódy modulu rozpoznávače řeč pro UniMRCP.